Задача A. Пары

Для каждого разбиения на две пары четырёх чисел надо просуммировать произведения каждой пары и выбрать из них максимум. Чтобы перебрать все варианты разбиения на две пары, можно было использовать циклы или перебрать их вручную, так как всего таких вариантов 3.

Задача B. Загадочное уравнение

Постановка задачи. Дано уравнение $x + y + xy = n$. Найти количество пар целых неотрицательных чисел x и y, которые являются решением этого уравнения.

Идея решения. Заметим, что многочлен $x + y + xy$ симметричен относительно своих переменных. Это значит, что если какая-то пара $x = a$ и $y = b$ является решением уравнения, то пара $x = b$ и $y = a$ также является решением данного уравнения. Тогда, подобно алгоритму проверки числа на простоту, переберем все x не превосходящие $\sqrt{n}$.

Для каждого x восстановим y: $y = (n - x) / (x + 1)$. Если y целое, добавим пары (x, y) и (y, x) в ответ.

Не забудем, что если $x = y$, то нужно добавить только одну пару.

Задача C. K-перестановки

Заметим, что перестановок из девяти чисел немного, всего 9! = 362880. Поэтому можно просто перебрать все перестановки и из n чисел и проверить каждую, не является ли она k-перестановкой.

Приведем фрагмента программы на языке Паскаль.

```
procedure go(pos : integer);
var
  i : integer;
begin
  if (pos = n + 1) then begin
    for i := 1 to n - 1 do begin
      if abs(p[i] - p[i + 1]) > k then begin
        exit;
      end;
    end;
    inc(ans);
    exit;
  end;
  for i := 1 to n do begin
    if (not w[i]) then begin
```

```
      w[i] := true;
      p[pos] := i;
      go(pos + 1);
      w[i] := false;
    end;
  end;
end;
```

Процедура go(pos) перебирает все возможные еще не использованные числа, которые можно поставить на позицию pos в перестановке p, если мы знаем все предыдущие числа. Для того, чтобы эффективно проверять, было использовано число или нет, поддерживается булевый массив w, причем w[i] = true если i уже использовано и false иначе. Если pos > n, то мы построили какую-то перестановку, осталось проверить, что она является k - перестановкой. Если это так, то увеличиваем ответ переменную ans.

Задача D. Сумма

Будем решать динамическим программированием и считать следующую величину: d[какая сумма получена][последнее слагаемое, которое мы брали].

Пересчёт следующий:

$$d[i][j] = \begin{cases} d[i][j-1], j - \text{чётное}, \\ d[i][j-i] + d[i-j][j], j - \text{нечётное}. \end{cases}$$

Ответ будет храниться в d[n][n]. Для хранения чисел в массиве d необходимо использовать 64-х битный целочисленный тип.

Задача E. Игра в слова

Как можно понять из условия задачи, необходимо проверить вхождение каждого слова из списка как подстроки в каждую строку и в каждый столбец таблицы. Для этого можно перебирать все возможные позиции начала слова в таблице и направление написания, а затем посимвольно сравнивать символы искомого слова с символами в таблице. Такой алгоритм требует $O(n \cdot m \cdot k \cdot maxlen)$, где $maxlen$ – максимальная длина слова. При ограничениях, указанных в условии, на максимальном по размеру тесте этому алгоритму требуется менее 15 миллионов операций, что укладывается в ограничение по времени.

Фрагмент программы на языке Паскаль:

```
    for w:= 1 to k do begin
```

```
      readln(s);
      found:= false;
      for x:= 1 to n do
        for y:= 1 to m do begin
          i:= 0;
          while (x+i<=n) and (i<length(s)) and
(s[i+1]=a[x+i][y]) do
            inc(i);
          if (i=length(s)) then
            found:= true;
          i:= 0;
          while (y+i<=m) and (i<length(s)) and
(s[i+1]=a[x][y+i]) do
            inc(i);
          if (i=length(s)) then
            found:=true;
          end;
        if (found) then
          writeln('YES ')
        else
          writeln('NO');
        end;
```

Здесь a: array [1..50] of string — массив строк, содержащий таблицу.

Также задача поиска образца в строке может быть решена за линейное время от суммы их длин алгоритмом Кнута-Морриса-Пратта.

Применяя данный алгоритм ко всем парам слово-строка и слово-столбец, получаем алгоритм, работающий за $O(k \cdot (n \cdot (maxlen + m) + m \cdot (maxlen + n)) = O(k \cdot (n + m) \cdot maxlen + k \cdot n \cdot m)$. Это решение задачи более эффективно по времени.

Фрагмент программы на языке Паскаль:

```
    readln(n, m);
    for i:=1 to n do
      readln(a[i]);
    readln(k);
    for x:=1 to k do begin
      readln(s);
      found:=false;
      j:=0;
      for i:=2 to length(s) do begin
        while(j>0) and (s[i]<>s[j+1]) do
          j:=p[j];
```

```
      if (s[i]=s[j+1]) then
        inc(j);
      p[i]:=j;
    end;
    for y:=1 to n do begin
      j:=0;
      for i:=1 to length(a[y]) do begin
        while (j>0) and (a[y][i]<>s[j+1]) do
          j:=p[j];
        if (a[y][i]=s[j+1]) then
          inc(j);
        if (j>=length(s)) then
          found:=true;
        end;
      end;
      for y:=1 to m do begin
        j:=0;
        for i:=1 to n do begin
          while (j>0) and (a[i][y]<>s[j+1]) do
            j:=p[j];
          if (a[i][y]=s[j+1]) then
            inc(j);
          if (j>=length(s)) then
            found:=true;
        end;
      end;
      if (found) then
        writeln('YES')
      else
        writeln('NO');
    end;
```